Conservative Behavioural Modelling in SystemC-AMS

Conservative Behavioural Modelling in SystemC-AMS

Sara Vinco
Dept. of Control & Computer Eng.
Politecnico di Torino,
Torino, Italy
sara.vinco@polito.it

Michele Lora
Dept. of Computer Science,
Università degli Studi di Verona
Verona, Italy
michele.lora@univr.it

Mark Zwolinski
Dept. of Electronics & Computer Science,
University of Southampton
Southampton, UK
mz@ecs.soton.ac.uk

***Abstract*—SystemC has recently been extended with the Analogue and Mixed Signal (AMS) library, with the ultimate goal of providing simulation support to analogue electronics and continuous time behaviours. SystemC-AMS allows modelling of systems that are either conservative and extremely low level or continuous time and behavioural, which is limited compared to other AMS HDLs. This work faces up this challenge, by extending SystemC-AMS support to a new level of abstraction, called *Analogue Behavioural Modelling* (ABM), covering models that are both behavioural and conservative. This leads to a methodology that uses SystemC-AMS constructs in a novel way. Full automation of the methodology allows proof of its effectiveness both in terms of accuracy and simulation performance, and application of the overall approach to a complex industrial Micro Electro-Mechanical System (MEMS) case study. The effectiveness of the proposed approach is further highlighted in the context of virtual platforms for smart systems, as adopting a C++-based language for MEMS simulation reduces the simulation time by about 2x, thus enhancing the design and integration flow.**

I. Introduction

SystemC has long been considered the reference language for electronic system-level design, as it supports both HW and SW and the integration of multiple levels of abstraction, including RTL and transactional level [1]. However, the increasing presence, in embedded systems, of analogue components and MEMS limits the generality of SystemC [2]. Indeed, this type of component requires the support of continuous time and conservative behaviours, which cannot be modelled with a discrete event simulator.

In response to this, Accellera has standardized the SystemC-AMS extension [3]. SystemC-AMS provides a number of predefined levels of abstraction that reproduce linear continuous time models with different degrees of accuracy and adherence to physical behaviours. Unfortunately, such abstraction levels (briefly outlined in Figure 1) do not model all types of analogue models. The *Linear Signal Flow* (LSF) level of abstraction focuses on behavioural, continuous time systems, but it does not support the modelling of conservative systems. On the other hand, *Electrical Linear Network* (ELN) is conservative, but it does not support behavioural models. Furthermore, SystemC-AMS does not yet support non-linear modelling [9] and it can not therefore be considered a replacement for SPICE [4] or Verilog-AMS [5].

The resulting gap between ELN and LSF thus misses descriptions that are both behavioural and conservative, that are commonly used for the modelling of MEMS and other analogue components [6], [7]. The limited flexibility of SystemC-AMS forces designers to adopt other AMS HDLs (*e.g.*, Verilog-AMS) for modelling this kind of component, thus reducing the applicability of SystemC-AMS.

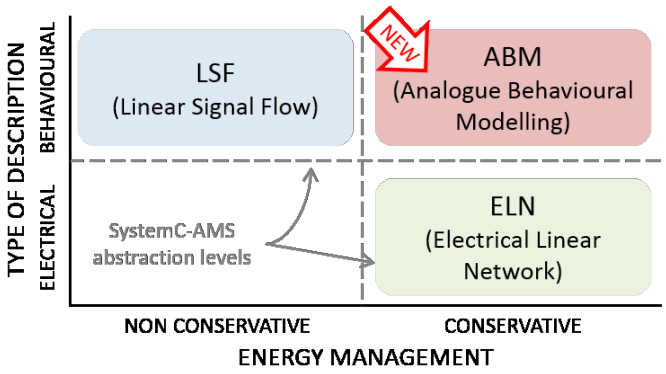

Fig. 1. Design space covered by the proposed approach *w.r.t.* SystemC-AMS. The methodology bridges the gap between LSF and ELN, by targeting behavioural and conservative descriptions (*ABM*).

The key idea of this work is to bridge the gap between LSF and ELN, to represent models that are both behavioural and conservative in SystemC-AMS. This new level of abstraction, called *Analogue Behavioural Modelling* (ABM), is demonstrated with a sound methodology that exploits existing SystemC-AMS constructs. The goal is to show that SystemC, with its AMS extensions, can be used as a general embedded system modelling and simulation framework even in the presence of analogue circuitry and MEMS. It is important to note that this paper does not define new SystemC-AMS libraries, but it rather uses ELN primitives in an innovative way.

The main contributions of this paper are:

- *identification of the ABM level of abstraction*, necessary for overall embedded system simulation in a SystemC based environment;
- definition of a *sound methodology* for modelling ABM components in SystemC-AMS, by using existing primitives in a novel way;
- *validation of the ABM level against Verilog-AMS*, to show that those Verilog-AMS descriptions that do not fall into the ELN and LSF domains can now be correctly represented in SystemC-AMS;

This work has been partially supported by the European project SMAC FP7-ICT-2011-7-288827.

- *automation of the proposed methodology* by automatically converting Verilog-AMS models into ABM SystemC-AMS models. This simplifies the application of the methodology to complex industrial case studies.

As a side effect of the proposed methodology, Verilog-AMS models can be automatically converted into SystemC-AMS code for easy integration into a virtual platform including analogue models. This avoids the use of CPU intensive co-simulation frameworks, thus noticeably speeding up the simulation of a virtual platform.

The paper is organized as follows. Section II provides the necessary background on Verilog-AMS and SystemC-AMS. Sections III and IV focus on the proposed methodology. Finally, Section V applies the proposed approach to an industrial case study and Section VI draws some conclusions.

II. Background

A. Verilog-AMS

Verilog-AMS is one of the most widely used languages for analogue and continuous time modelling [5]. In Verilog-AMS a circuit is modelled as an abstract graph of nodes (that can also be used for external connectivity) connected by branches [8]. System state is defined in terms of voltages (`V()`) and currents (`I()`) associated with nodes and branches. Relationships between nodes are modelled with algebraic and differential equations, called *simultaneous statements*. The *contribution operator* `<+` models a simultaneous statement summing multiple contributions to the branch current (or voltage) as a function of all branch voltages (or currents). The function is realized as a sequence of mathematical expressions, including time derivatives and integrals.

Conservative modelling is imposed by the requirement that the sum of currents leaving any node must be equal to zero at any time (thus reflecting Kirchhoff's laws). This condition is managed by the internal solver of the Verilog-AMS simulator, and thus it must not be explicitly modelled by the designer.

The simulator internal solver uses the simultaneous statements and conservative conditions to build a system matrix. Equations are thus solved iteratively at discrete time steps to determine system state over time.

B. SystemC-AMS

SystemC-AMS is the extension of the SystemC framework for modelling analogue and mixed-signal systems [3]. Its role is to provide a higher level view of mixed-signal and analogue systems, to allow early simulation and validation of the overall system. For this reason, SystemC-AMS supports only linear and time-invariant descriptions, and is incapable of solving non-linear functions [9].

To cover a wide variety of domains, SystemC-AMS provides three different abstraction levels, supporting different communication styles and representations *w.r.t.* the physical domain. *Timed Data-Flow* (TDF) models are scheduled statically by considering their producer-consumer dependencies in the discrete time domain. *Linear Signal Flow* (LSF) supports the modelling of continuous time through a library of pre-defined primitive modules (*e.g.*, integration, delay), each associated with a linear equation. Finally, the *Electrical Linear Network* (ELN) level models electrical networks through

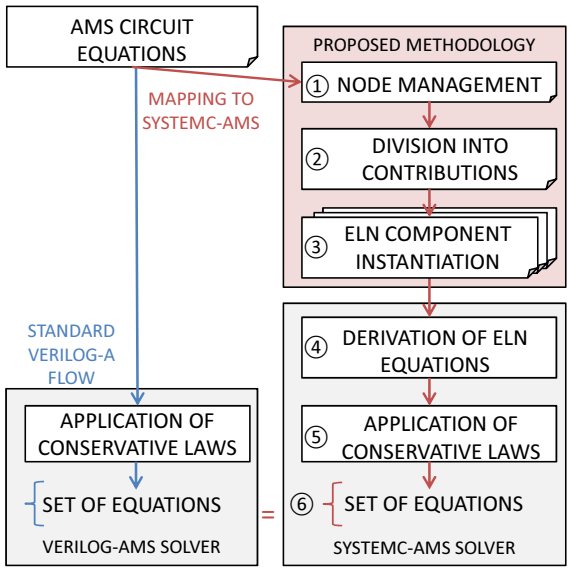

Fig. 2. Overview of the proposed methodology.

the instantiation of predefined primitives, *e.g.*, resistors or capacitors, associated with electrical equations. A *SystemC-AMS internal solver* analyses the ELN and LSF components to derive the equations modelling system behaviour, that are solved to determine system state at any simulation time.

The main difference between LSF and ELN is in the adherence to physical laws. LSF is *non-conservative* and it expresses behaviours as directed flows of continuous-time signals or quantities. On the contrary, ELN is *conservative*, *i.e.*, the derived set of equations is extended by the internal solver with the conservation laws.

III. Methodology Overview

The goal of the proposed approach is to prove that conservative and behavioural descriptions can be modelled in SystemC-AMS. To this extent, the starting point of the methodology is a Verilog-AMS behavioural description, made of a set of simultaneous statements that assemble voltages or currents to describe the state of the electrical circuit nodes. Due to the limitations of SystemC-AMS, the models are strictly linear and time-invariant.

The standard Verilog-AMS simulation flow is depicted on the left-hand side of Figure 2. The Verilog-AMS internal solver takes the simultaneous statements as the input and derives both the user defined equations and the conservative ones. The resulting equation set is used to build the numerical matrices that determine the system state.

The methodology to convert the Verilog-AMS code into SystemC-AMS is based on the reproduction of the final equation set in the SystemC-AMS environment, through the flow depicted on the right-hand side of Figure 2. First of all, Verilog-AMS nodes are mapped to SystemC-AMS nodes (①), and Verilog-AMS simultaneous statements are divided into basic contributions (②). Then, each contribution is mapped to a basic SystemC-AMS ELN element, where the equation associated with the ELN module is the same as the original Verilog-AMS contribution (③). The methodology determines how to connect the ELN modules (*i.e.*, in parallel or in series),

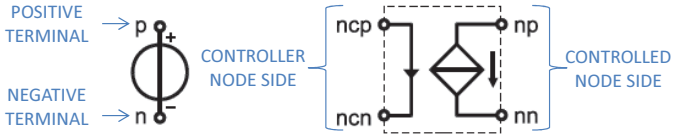

Fig. 3. ELN terminology applied to an independent source (left) and to a controlled source (right).

so that the bindings describe the same relationship between voltages and currents as the original Verilog-AMS simultaneous statement. The ELN system is then managed by the SystemC-AMS internal solver, that builds the corresponding equations (④) and adds conservative laws (⑤). The resulting equation system will thus reflect the Verilog-AMS one (⑥).

The choice of the ELN model of computation allows us to delegate the application of conservation laws to the internal solver. This is an important feature, as adding conservative laws implies reconstructing the circuit topology from the AMS equations, which can be far from trivial.

It is important to note that both the basis of the methodology and the correctness of the proposed approach lie in the construction of the same equation set, that is then solved similarly by the Verilog-AMS and SystemC-AMS AD solvers.

IV. Methodology

The following sections detail the proposed methodology. This work focuses on the construction of the ELN system (steps ① to ③ in Figure 2). The remaining steps (*i.e.*, the bottom box on the right-hand side of Figure 2) are automatically performed by the SystemC-AMS internal solver. The visual representation of ELN modules adopted in the following Figures is as defined by the SystemC-AMS standard [3].

A. The ABM abstraction level

The ABM abstraction level comprises descriptions mixing characteristics typical of digital behavioural models and of electrical conservative ones. ABM models are *behavioural* in that they do not directly reflect a HW or circuit implementation, but they are rather used in the design process to simulate a certain component's behaviour. At the same time, ABM models are *conservative* as they adopt circuit elements and constructs (*e.g.*, voltage and current values at circuit nodes), and thus abide by conservation laws.

These characteristics do not fit in any of the SystemC-AMS abstraction levels. Nonetheless, they are widely supported by other AMS HDLs for the design of components such as MEMS and analogue circuitry [6], [7]. It is thus necessary to extend SystemC-AMS, to improve its coverage and effectiveness. To avoid the burden of implementing a new SystemC-AMS abstraction level (and thus new classes and libraries), this work proposes a methodology that reduces ABM descriptions, modelled in other AMS HDLs, to SystemC-AMS ELN constructs. This guarantees the correctness of the underlying synchronization and solving mechanisms, and it preserves compatibility with any SystemC-AMS description.

B. ELN terminology

ELN modules have a standard interface made up of a positive terminal (`p` port) and a negative terminal (`n` port) for each contributing circuit node (left hand side of Figure 3). When an ELN module is controlled by any circuit node, the interface is as depicted on the right hand side of Figure 3. In detail, the interface has a source side (i.e, the result of the ELN module, here called the *controlled node*, with a positive terminal `np` and a negative terminal `nn`) and a control node side (*i.e.*, the input of the primitive module, here called the *controller node*, with a positive terminal `ncp` and a negative terminal `ncn`).

C. Circuit node management

The first declaration added to the SystemC-AMS code is the instantiation of ground, declared as a node of type `sca_node_ref`. Verilog-AMS nodes are mapped to SystemC-AMS ELN circuit nodes (of type `sca_node`). Each node is then connected to ground through a 1 GΩ resistor, by using the ELN `sca_r` primitive. This is identical to the Gmin conductance that SPICE automatically inserts between each node and ground, and it helps to ensure that the final equation set can be solved.

D. Division into contributions

SystemC-AMS is less expressive than Verilog-AMS, *i.e.*, it supports a more restricted range of constructs and ELN models can be composed only of instances of the predefined primitives [7], [10]. *E.g.*, in SystemC-AMS a voltage value can be controlled only by one voltage contribution, while Verilog-AMS allows any number of contributions. Thus, any Verilog-AMS simultaneous statement must be reproduced by connecting a number of ELN elements.

Given a Verilog-AMS description, the methodology identifies the contributions comprising each simultaneous statement by finding the largest sub-equation that can be represented by a single ELN object. In linear and time-invariant descriptions this corresponds to breaking the equation into the single addends.

E. Mapping to ELN components

The remainder of this section presents how a set of template equations are mapped to ELN primitives to model their individual contributions and how such primitives are connected.

*1) **Voltage sources**:*
Voltage source Verilog-AMS equations use a number of contributions to assign a voltage level to a circuit node. Contributions can be of three main types: independent, voltage-controlled and current-controlled. A complete example of a voltage source equation is shown in Figure 4.

Independent voltage sources assign a numerical voltage value, and they correspond to contributions like:

```
V(a) <+ 8.01
```

(*i.e.*, contribution 3 in Figure 4). They are implemented by using a `sca_vsource` ELN module, where the voltage value is an instantiation parameter (*i.e.*, +8.01). The module interface has only a positive terminal, connected to the controlled node (`a`), and a negative terminal, connected to ground (`gnd`).

A *voltage controlled voltage source* is a voltage source whose value depends on the voltage level at a certain circuit node. An example is contribution 1 in Figure 4:

```
V(a) <+ +4.02 V(b)
```

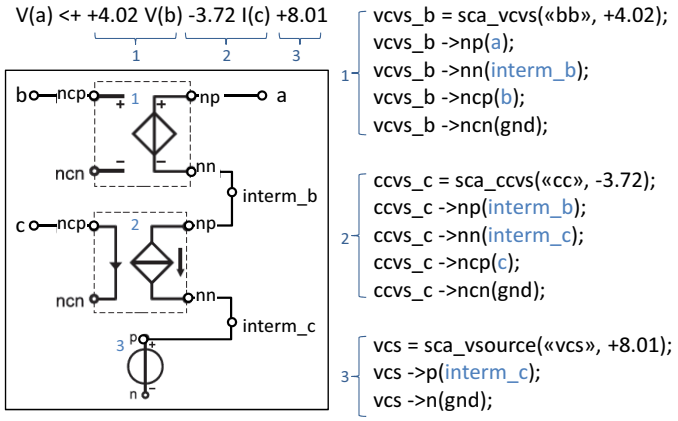

Fig. 4. Example of voltage source equation (top left) with the corresponding SystemC-AMS code (right) and ELN module connection (bottom left). Non-connected terminals are connected to ground.

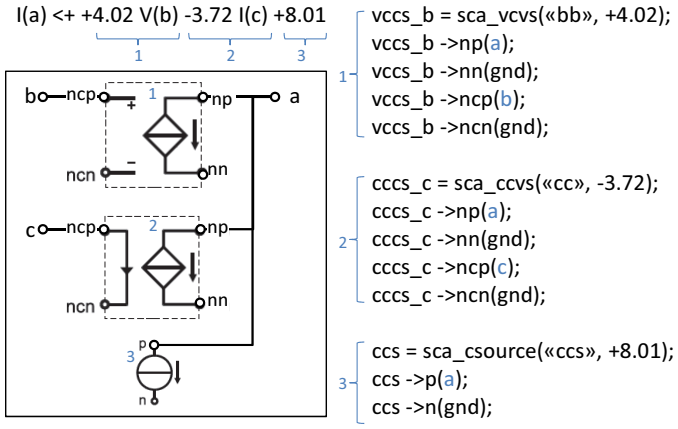

Fig. 5. Example of current source equation (top left) with the corresponding SystemC-AMS code (right) and ELN module connection (bottom left). Non-connected terminals are connected to ground.

It is implemented by using the `sca_vcvs` ELN module, where the scaling factor is an instantiation parameter (*i.e.*, +4.02). The module interface has a controlling node side (whose positive terminal is connected to `b`) and a controlled node side (whose positive terminal is connected to `a`). Negative controlling terminals are connected to ground.

A *current controlled voltage source* describes a voltage source whose value depends on the current through a certain circuit branch. An example is contribution 2 in Figure 4:

```
V(a) <+ -3.72 I(c).
```

Such contributions are implemented by using the `sca_ccvs` ELN module, connected to node `a` as the controlled node and to node `c` as the controlling node.

If a Verilog-AMS voltage source equation is made of more than one contribution, SystemC-AMS instances are *connected in series*. This is achieved by creating intermediate nodes that connect the `nn` terminal of a primitive with the `np` terminal of the next primitive. In this way, voltage values add up and any new contribution is added in series with the former ones. In Figure 4, this is achieved by introducing nodes `interm_b` (that connects contributions 1 and 2) and `interm_c` (that connects contributions 2 and 3).

*2) **Current sources**:*
Current source Verilog-AMS equations are the complement of voltage source equations for current, *i.e.*, use a number of contributions to assign an input current to a circuit node. A complete example is shown in Figure 5.

An *independent current source* assigns a numerical current value and is implemented by using the `sca_isource` ELN module (contribution 3 in Figure 5). A *voltage controlled current source* defines a current source whose value depends on the voltage level at a certain circuit node (contribution 1 in Figure 5). These kinds of contributions are mapped to `sca_vccs` ELN modules. Finally, a *current controlled current source* describes a current source whose value depends on the current flowing through a certain circuit branch (contribution 2 in Figure 5). Such contributions are implemented by using `sca_cccs` ELN modules.

If a Verilog-AMS current source equation is made up of more than one contribution, SystemC-AMS instances are *connected in parallel*. The `ncp` terminal of each module is connected to the controlling node (`b` and `c`) and the `np` (or `p`) terminal is connected to the controlled node (`a`). In this way, the voltage is the same across all involved circuit branches and the current is summed at the controlled node `a`.

*3) **Differential constructs**:*
Differential contributions are more complex than voltage or current ones, as they model a derivative (or integrative) relationship between the current or voltage of two separate circuit nodes. SystemC-AMS, on the other hand, restricts differential behaviours to dependencies on single network nodes, through the adoption of capacitors (`sca_c` ELN module) or inductors (`sca_l`). To overcome this limitation, it is necessary to introduce an intermediate node that has no physical correspondence in the circuit, but that is rather used for describing the differential dependence.

a) Derivative contributions: Derivative contributions describe a derivative dependency between the voltage or current quantities of two different circuit nodes, *e.g.*:

```
I(a) <+ ddt(+4.02 V(b))
```

To overcome SystemC-AMS limitations, the original Verilog-AMS differential contribution is divided into three separate contributions, as outlined in Figure 6. The circuit is enhanced with a new node (`interm`), used to represent the derivative dependency as an inductor (implemented as an instance of a `sca_l` ELN module). This adds the following equation (equation 2 in Figure 6):

```
V(interm) = ddt(I(interm))
```

Then, two equations are added to bind the values in `a` and `b` to the current and voltage values in the new node `interm`. In Figure 6, node `a` is modelled as a current source dependent on the voltage in `interm` (the dependency is implemented as an instance of `sca_vccs`). This adds equation 3:

```
I(a) = +4.02 V (interm)
```

The current through `interm` is controlled by the voltage at `b` (the dependency is implemented as an instance of `sca_vccs`). This adds equation 1:

```
I(interm)= V(b)
```

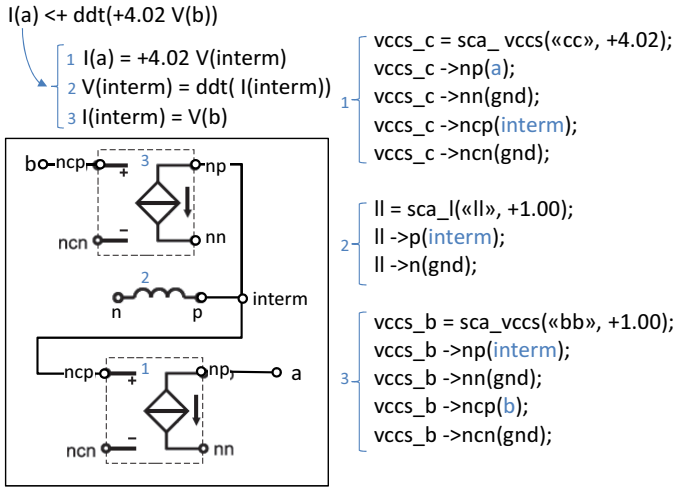

Fig. 6. Example of derivative equation with corresponding individual contributions and equations (top left), SystemC-AMS code (bottom left) and ELN module connection (right).

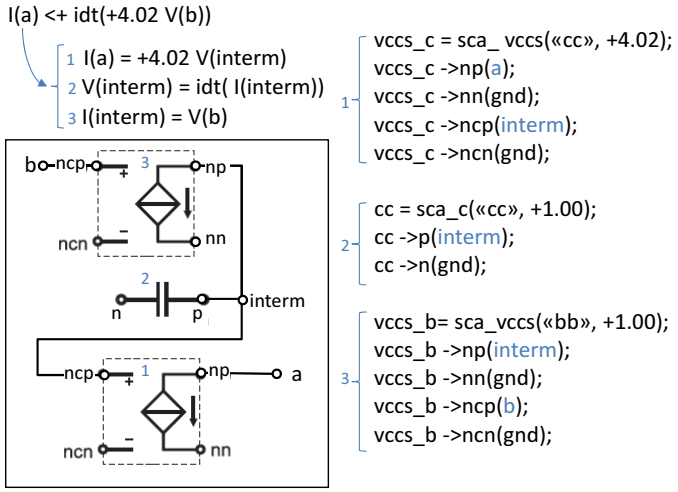

Fig. 7. Example of integrative equation with corresponding individual contributions and equations (top left), SystemC-AMS code (bottom left) and ELN module connection (right).

The resulting system of equations will thus reconstruct the original dependency between nodes:

```
I(a) = +4.02V(interm)= +4.02ddt(I(interm))
                      = +4.02ddt(V(b))
```

The combination of voltage or current sources strictly depends on the starting simultaneous statement. However, the same approach can be easily extended to any combination of voltage/current dependencies.

The resulting SystemC-AMS sub-system is depicted on the left hand side of Figure 6. The sub-system can be further connected to other ELN models if it is included in more complex simultaneous statements, by adopting either parallel or series compositions.

b) Integrative contributions: Integrative contributions describe an integrative dependency between the voltage or current quantities of two different circuit nodes, *e.g.*:

```
I(a) <+ idt(+4.02 V(b))
```

The approach used to overcome SystemC-AMS limitations reflects the solution outlined for derivative contributions (Figure 7). The circuit is extended with a new node (`interm`), used to represent the integrative dependency as a capacitor (implemented as an instance of the `sca_c` ELN module). This adds to the system equation 2:

```
V(interm) = idt(I(interm))
```

Then, two equations are added to bind the values in `a` and `b` to the current and voltage values in the new node `interm`, similarly to the solution proposed for derivative contributions:

```
I(a) = +4.02 V(interm)
I(interm) = V(b)
```

The resulting set of equations will thus reproduce the original dependency between nodes:

```
I(a) = +4.02V(interm)= +4.02idt( I(interm))
                     = +4.02idt( V(b))
```

Once again, the combination of voltage or current sources strictly depends on the starting simultaneous statement. However, the same approach can be easily extended to any combination of voltage/current dependencies.

The resulting SystemC-AMS sub-system is depicted on the left hand side of Figure 7. The sub-system can be further connected to other ELN models if it is included in more complex simultaneous statements.

V. Experimental results

This section demonstrates the effectiveness of the proposed approach in terms of accuracy and simulation time. All experiments were evaluated on an i7 3.2GHz processor with 16GB RAM, running Ubuntu 12.04. Verilog-AMS descriptions have been simulated using Mentor's Questa 13.1 simulator [11].

A. Methodology automation

Manual application of the proposed methodology is a tedious error-prone process, whose application to industrial case studies could be extremely tricky. For this reason, we implemented the automatic tool *ABACuS* (Analogue BehAvioural Conservative Systemc-ams). *ABACuS* leverages the academic licence version of HIFSuite to ease the conversion process [12]. Verilog-AMS descriptions are analysed and translated into the HIFSuite internal format (HIF). The code generated at this point is a tree-structured XML-like representation of the original code. *ABACuS* applies a number of processing steps to the HIF description to automate the methodology, including contribution identification and construction of the ELN system. This leads a new HIF description, containing the instantiation and connection of the corresponding ELN primitives. The HIF description is then converted to SystemC-AMS by means of the HIFSuite *hif2sc* back-end tool.

B. Methodology validation

The first step to validate the proposed methodology is to prove the correctness of the mapping of single contributions detailed in Section IV. Validation focuses on 7 case studies, each targeting a single type of contributions. The case studies

TABLE I. VALIDATION OF THE MAPPING OF EACH TYPE OF CONTRIBUTION TO ELN CONSTRUCTS.

Case study	Target contribution	Simulation time (s)	Error (%)
1	$\mathtt{V}(out) <+\ k_1\mathtt{V}(in_1)+\ldots+k_n\mathtt{V}(in_n)+c$	14.01	0.062
2	$\mathtt{V}(out) <+\ k_1\mathtt{I}(in_1)+\ldots+k_n\mathtt{I}(in_n)+c$	14.17	0.062
3	$\mathtt{I}(out) <+\ k_1\mathtt{V}(in_1)+\ldots+k_n\mathtt{V}(in_n)+c$	14.11	0.062
4	$\mathtt{I}(out) <+\ k_1\mathtt{I}(in_1)+\ldots+k_n\mathtt{I}(in_n)+c$	14.05	0.062
5	$\mathtt{I}(out) <+\ \mathtt{k\ ddt}(\mathtt{V}(in_1))\ +\ c$	14.61	0.006
6	$\mathtt{V}(out) <+\ \mathtt{k\ ddt}(\mathtt{V}(in_1))\ +\ c$	14.53	0.006
7	$\mathtt{I}(out) <+\ \mathtt{k\ idt}(\mathtt{V}(in_1))\ +\ c$	14.25	0.028

were implemented in Verilog-AMS and have been converted to SystemC-AMS via *ABACuS*. The main characteristics of each case study are reported in Table I, in terms of target contribution type and simulation time. Case studies are fed with sinusoidal inputs with 100Hz frequency, so that the outputs can be easily controlled and compared *w.r.t.* the expected result. In particular, the system of equations described using Verilog-AMS has been computed symbolically and solved for every instant of time corresponding to a sample of the SystemC-AMS execution. The SystemC-AMS simulation is run with a 1us timestep, while the execution is sampled with a period of 10us. The table shows that the error *w.r.t.* the expected behaviour (computed as point-to-point difference) is lower than 0.1%. This error is due to the precision issues of the numerical algorithms used by the simulator to perform continuous time simulation. Thus, the table highlights the accuracy in the mapping of all types of contributions.

Simulation times in Table I refer to the amount of time needed to simulate 1 second of the design. For all the cases depicted in table, the time needed for simulating the starting Verilog-AMS matches that needed for the SystemC-AMS simulation. This is due to the fact that both the solvers (*i.e.*, SystemC-AMS and Questa) are solving the same set of equations, as advocated in Section III.

The similar simulation times imply that the proposed translation to SystemC-AMS does not provide any simulation speed up, as both the simulators solve the same set of equations with similar strategies. However, Sections V-D and V-E will highlight the effectiveness when handling more complex designs, including mixed analogue and discrete descriptions.

C. Methodology scalability

Table II proves the scalability of the proposed approach by estimating the impact of the adopted timestep on the accuracy of the generated code. The table focuses on case study 1 (*i.e.*, $\mathtt{V}(out) <+\ k_1\mathtt{V}(in_1)+\ldots+k_n\mathtt{V}(in_n)+c$), but similar results apply also to the other case studies.

The SystemC-AMS code is stimulated with three different sinusoidal inputs, with increasing maximum input frequency. For each input, we simulated the code with different time steps, ranging from 0.5us up to 10us.

The simulation time decreases linearly with the length of the time step, with a speedup of approx. 20x between adopting a timestep of 10us *w.r.t.* a timestep of 0.5us. At the same time, accuracy is preserved, as the average error is below 1% even when the timestep is increased tenfold. It is important to note that the error varies depending not only on the adopted timestep, but also on the frequency of the sinusoidal inputs. As a result, the highest accuracy (0.003%)

TABLE II. SCALABILITY OF THE PROPOSED METHODOLOGY *w.r.t.* THE SIMULATION TIMESTEP.

Input frequency	Adopted timestep	Avg. error (%)	Simulation time (s)
10 Hz	0.5 us	0.003	28.09
	1.0 us	0.006	14.15
	10.0 us	0.063	1.42
100 Hz	0.5 us	0.032	27.97
	1.0 us	0.062	14.01
	10.0 us	0.628	1.42
1 KHz	0.5 us	0.314	28.05
	1.0 us	0.607	13.92
	10.0 us	6.601	1.39

is reached with the 10Hz input and timestep of 0.5us. On the contrary, the configuration with the 1KHz input and the 10us timestep performs worse than the others, with an average error of 6.601%, because the adopted time step is too coarse for the input frequency. These considerations highlight the importance of choosing a suitable timestep for the simulation, but also that the generated SystemC-AMS code allows us to determine accuracy/simulation speed trade offs.

D. The MEMS accelerometer

In order to prove the effectiveness of the overall methodology on more complex designs, we applied the overall approach to a complex industrial case study, developed in the context of an industry-funded project. The case study is a *2-dimensional MEMS accelerometer* implemented in Verilog-AMS by means of the MEMS design platform MEMS+, that supports automatic Verilog-AMS code generation [6], starting from 3-dimensional physical models as the one depicted in Figure 9. The choice of a MEMS design was guided by the consideration that MEMS behavioural modelling is based on differential and algebraic equations [7], thus following the Verilog-AMS structure assumed in this paper. Table III reports the main characteristics of the MEMS design, both in terms of simultaneous statements and of types of contributions.

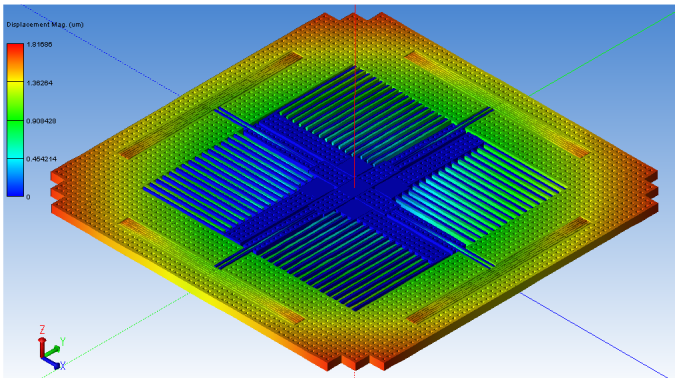

Fig. 8. 3-dimensional model of the accelerometer in the MEMS+ design simulator.

Table IV shows the results of the application of *ABACuS* to the MEMS design. The table shows the number of lines of code of the resulting SystemC-AMS implementation, the number of added nodes and of instances of SystemC-AMS primitives. The number of lines of codes is increased tenfold (precisely, 11.12x), as the SystemC-AMS generated by the methodology is more verbose than Verilog-AMS. Each contribution requires

TABLE III. CHARACTERISTICS OF THE ORIGINAL VERILOG-AMS MEMS DESIGN.

Lines of code		89
Equations	Voltage sources	10
	Current sources	15
Node declarations	Interface	14
	Internal	14
Contributions	Independent	4
	Voltage	59
	Current	0
	Derivative	12
	Integrative	0

TABLE IV. CHARACTERISTICS OF THE GENERATED SYSTEMC-AMS MEMS DESIGN.

Lines of code		1,474
Added node declarations		12
SystemC-AMS primitive instantiations	`sca_r`	93
	`sca_vsource`	4
	`sca_vcvs`	32
	`sca_ccvs`	0
	`sca_csource`	0
	`sca_vccs`	48
	`sca_cccs`	0
	`sca_l`	12
	`sca_c`	0

the instantiation of the ELN primitive, plus the corresponding explicit port binding. Furthermore, the number of ELN primitives is higher than the number of Verilog-AMS contributions. This is due to the presence of 12 derivative contributions in the original Verilog-AMS code. Each such contribution determines the instantiation of three ELN primitives (as explained in Section IV-E3). As a result, of the 188 resulting SystemC-AMS ELN instances:

- 93 correspond to resistors added to connect each SystemC-AMS node to ground;
- 59 correspond to voltage source contributions;
- 36 are generated by the 12 derivative constructs, that determine also the declaration of 12 additional internal nodes.

The numbers highlight that *ABACuS* strictly follows the presented methodology, in particular:

- one resistor is added for each circuit node;
- each non-derivative contribution determines the addition of one ELN primitive instance;
- each derivative contribution generates three ELN primitive instances.

Fast code generation is a major advantage of the proposed approach. Table V highlights that code generation is almost instantaneous (17.48s overall), and that most of the effort in spent in the HIFSuite conversions (55%). The most costly step of *ABACuS* execution lies in the mapping from Verilog-AMS contributions to ELN primitives and in their instantiation (37%). On the other hand, node management and the separation of Verilog-AMS equations into single contributions is almost immediate.

The generated code was validated by comparing its execution *w.r.t.* the original Verilog-AMS code. SystemC-AMS simulation was run by adopting the same input stimula of

TABLE V. CHARACTERISTICS OF THE EXECUTION OF *ABACuS* ON THE MEMS DESIGN.

Overall		17.48s
HIFSuite tools	Conversion to HIF	1.86s
	Conversion to SystemC-AMS	7.81s
ABACuS	Node management	0.94s
	Division into contributions	0.29s
	ELN component instantiations	6.58s

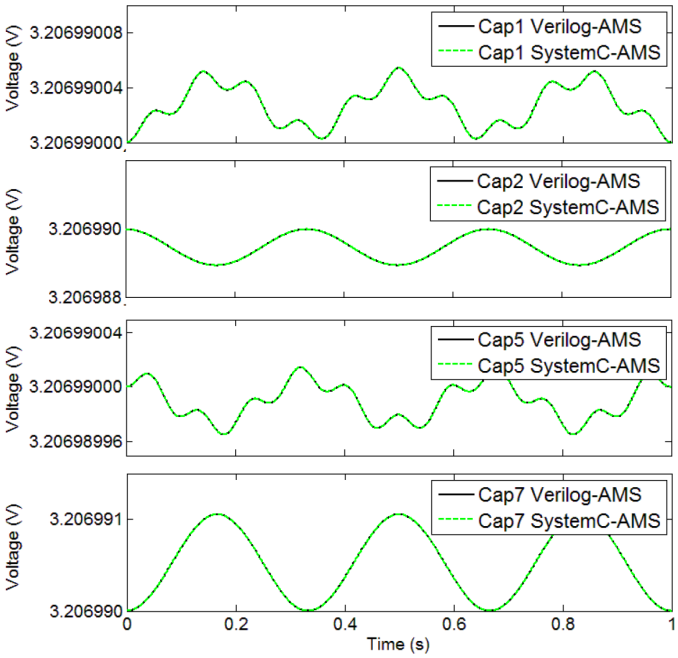

Fig. 9. Evolution of the MEMS outputs for Verilog-AMS (solid) and SystemC-AMS (dashed).

the Verilog-AMS implementation, and with a 1us timestep. SystemC-AMS proved to be slightly faster than the Verilog-AMS execution (28.02s and 33.72s, respectively). At the same time, the average error in the computation of the MEMS outputs is 0.02%. This confirms the visual accuracy evident from Figure 9, where the Verilog-AMS and SystemC-AMS curves are almost totally overlapping. The small error is due to the different management of time in the two simulators: SystemC-AMS adopts a fixed timestep, while Verilog-AMS can adapt the length of the timestep over time, thus reaching a higher accuracy. The low error rate highlights the effectiveness of the generated code, both in terms of accuracy and of simulation speed.

E. Effectiveness of the proposed approach

The most important advantage of modelling ABM models in SystemC-AMS lies in the ease of integration in more complex platforms and in the enhanced support for virtual platforms and system-level design, rather than in the pure accuracy or simulation speed.

The SystemC-AMS scheduler is an extension of the discrete-event SystemC scheduler, and it thus allows simultaneous simulation components belonging to heterogeneous domains. Furthermore, integration with a C++ system level description is eased, thus further removing computationally expensive interfaces and thus speeding up the simulation

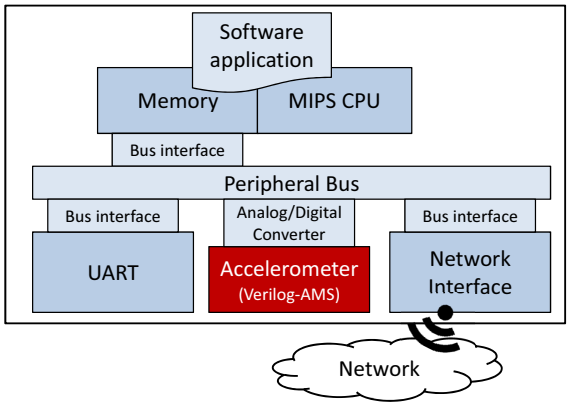

Fig. 10. Overview of the virtual platform containing the MEMS (*i.e.*, the Accelerometer) component. Digital components, implemented using SystemC, are colored in light blue. The MEMS (colored in red) is originally implemented in Verilog-AMS.

of mixed-signal systems. For these reasons, SystemC-based languages are a winning solution for the construction and validation of virtual platforms, and they are adopted by most of the currently available virtual platform environments [13]–[16]. Translating ABM models to SystemC-AMS thus allows their early validation, together with the interaction with other system components.

The MEMS accelerometer has been integrated into a virtual platform for smart systems. The structure of the platform is depicted in Figure 10. It includes (1) a 32-bit RISC processor (the *MIPS CPU*) executing (2) a *Software Application* elaborating data sensed by the accelerometer and stored in (3) a *Memory*. External communication is managed by a (4) Universal Asynchronous Receiver/ Transmitter (*UART*) and by a (5) *Network Interface*, used to send and receive data to and from other smart sensors. All system components, except the MEMS, are implemented in SystemC, and integrated in a virtual platform. Validating the integration of the original MEMS component in the overall system would thus require the construction of a simulation framework.

The first alternative to validate the integration of the MEMS component in the platform is to preserve the language heterogeneity, but within a single simulator. Thus, we adopted Questa [11], which handles both discrete-time and analogue descriptions, and that natively provides SPICE-based constructs to connect analogue and digital designs.

The second alternative is to adopt the methodology proposed in this paper to convert the MEMS design to SystemC-AMS, integrate it in the virtual platform and to run the overall system with the SystemC simulator.

Table VI reports the time needed to simulate 100ms of the real system execution, and it shows how the SystemC based simulation outperforms Questa (2.21x). This is mainly due to the heavy communication overhead induced by Questa to allow communication and synchronization between the discrete event and the Spice-base simulators used by Questa respectively for the SystemC and Verilog-AMS parts of the model. At the same time, SystemC-AMS provides a good level of accuracy (0.02%), thus constituting a valide alternative for early validation of the overall system and of the analogue-digital communication.

TABLE VI. SIMULATION TIME OF THE VIRTUAL PLATFORM BY PRESERVING THE LANGUAGE HETEROGENEITY AND MOVING TO SYSTEMC-AMS.

Languages	Simulator	Simulation time (s)
SystemC and Verilog-AMS	Questa	215.47
SystemC and SystemC-AMS	SystemC-AMS kernel	97.59

VI. CONCLUSIONS

The work described here proposes a methodology for representing models that are both conservative and behavioural in SystemC-AMS. We achieve this goal by adopting existing SystemC-AMS ELN primitives in a novel way. As a result, SystemC effectiveness is enhanced in the context of embedded system design, as it can cover a wider range of descriptions and components. Experimental results highlight the correctness of the proposed approach both on synthetic case studies, focusing on the single methodology steps, and on a complex industrial MEMS case study. Future work will focus on the identification of abstraction strategies to target the SystemC-AMS Timed Data Flow (TDF) level for improved simulation performance.

REFERENCES

[1] IEEE, "1666-2011 - IEEE Standard for Standard SystemC," 2011, standards.ieee.org/findstds/standard/1666-2011.html.

[2] R. Zafalon, "Smart system design: Industrial challenges and perspectives," in *Proc. of IEEE MDM*, 2013, p. 3.

[3] Accellera Systems Initiative, "SystemC-AMS and Design of Embedded Mixed-Signal Systems," 2013, accellera.org/activities/working-groups/systemc-ams.

[4] L. W. Nagel and D. O. Pederson, *SPICE: Simulation program with integrated circuit emphasis*. Electronics Research Laboratory, College of Engineering, University of California, 1973.

[5] Accellera Systems Initiative, "Verilog-AMS," 2014, accellera.org/downloads/standards/v-ams.

[6] Coventor, Inc., "MEMS+: MEMS Simulation Software," www.coventor.com/mems-solutions/products/mems.

[7] P. Schneider, C. Bayer, K. Einwich, and A. Kohler, "System level simulation - A core method for efficient design of MEMS and mechatronic systems," in *Proc. of IEEE SSD*, 2012, pp. 1–6.

[8] S. Mijalkovic, "Advanced circuit and device modeling with Verilog-A," in *Proc. of IEEE MIEL*, 2006, pp. 439–442.

[9] P. Hartmann, P. Reinkemeier, A. Rettberg, and W. Nebel, "Modelling control systems in SystemC-AMS – Benefits and limitations," in *Proc.of IEEE SOCC*, 2009, pp. 263–266.

[10] R. Narayanan, N. Abbasi, M. Zaki, G. A. Sammane, and S. Tahar, "On the simulation performance of contemporary AMS hardware description languages," in *Proc. of IEEE ICM*, 2008, pp. 361–364.

[11] Mentor Graphics, "Questa Advanced Simulator," www.mentor.com/products/fv/questa.

[12] N. Bombieri, G. Di Guglielmo, M. Ferrari, F. Fummi, G. Pravadelli, F. Stefanni, and A. Venturelli, "Hifsuite: tools for hdl code conversion and manipulation," *EURASIP Journal on Embedded Systems*, vol. 2010, pp. 4:1–4:20, Jan. 2010.

[13] Synopsys, "Platform architect," www.synopsys.com/Prototyping/ArchitectureDesign.

[14] Cadence, "Virtual System Platform," www.cadence.com/products/sd/virtual_system.

[15] Imperas Software, "OVP - Open Virtual Platforms," www.ovpworld.org.

[16] Mentor Graphics, "Vista Virtual Prototyping for SystemC/TLM 2.0 and QEMU ," www.mentor.com/esl/vista/virtual-prototyping.